


INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen
Oettingenstraße 67, D-80538 München

—————
Ludwig ———
Maximilians —
Universität ———
München ———



Pattern Queries for XML and Semistructured Data

(revised version of PMS-FB-2002-5)

François Bry and Sebastian Schaffert

Technical Report, Computer Science Institute, Munich, Germany
<http://www.pms.informatik.uni-muenchen.de/publikationen>
Forschungsbericht/Research Report PMS-FB-2003-1, January 2003

Pattern Queries for XML and Semistructured Data

Sacha Berger, François Bry and Sebastian Schaffert

Institute for Computer Science, University of Munich
<http://www.pms.informatik.uni-muenchen.de/>

Abstract. Query and transformation languages developed since the mid 90es for XML and semistructured data – e.g. XQuery, the precursors of XQuery, and XSLT – built upon a path-oriented node selection: A node in a data item is specified in terms of a root-to-node path in the manner of the file selection languages of operating systems. Constructs from regular expressions such as *, +, ?, and “wildcards” give rise to a flexible node retrieval from incompletely specified data items. This paper investigates an alternative approach to querying XML and semistructured data. A metaphor for this approach views queries as patterns, answers as data items matching the queries. Formally, an answer to a query is defined as a simulation [1] of an instance of the query in a data item. The basics of the query language Xcerpt are introduced and it is given a model-theoretic semantics. Recent results on an operational semantics are outlined. Ideas for a visual query language are finally sketched.

1 Introduction

Essential to semistructured data is the selection of data from incompletely specified data items. For such a data selection, a path language such as XPath [2] is convenient because it provides with regular expressions such as *, +, ?, and “wildcards” that give rise to a flexible node retrieval.

Query and transformation languages developed since the mid 90es for XML [2] and semistructured data – e.g. XQuery [2], the precursors of XQuery [3], and XSLT [2] – rely upon such a path-oriented selection. They use patterns (also called templates) for expressing how the selected data, expressed by paths, are re-arranged (or re-constructed) into new data items. Thus, such languages intertwine construct parts, i.e. the construction patterns, and query parts, i.e. path selectors.

This intertwining has some advantages: For simple query-construct requests, the approach is rather natural and results in an easily understandable code. However, intertwining construct and query parts also has drawbacks:

1. Query-construct requests involving a complex data retrieval might be confusing,
2. unnecessarily complex path selections, e.g. XPath expressions involving both forward and reverse axes, are possible [4],
3. in case of several path selections, the overall structure of the retrieved data items might be difficult to grasp for a human reader.

This paper addresses using patterns instead of paths for querying XML and semistructured data, an approach first proposed with UnQL [5] and further investigated e.g. with XMAS [6]. A metaphor for this approach is to see queries

as forms, answers as form fillings yielding database items. With this approach, patterns are used not only in construct expressions, for re-arranging the retrieved data into new data items, but also for data selection. Furthermore, this paper argues that the pattern approach allows for a more natural visualisation of query and construct expressions in a visual query language.

In the following, the basics of the query language Xcerpt, which is based on logic programming, are introduced. An answer to a query in this language is formalised as a simulation [1] of a ground instance of the query in a database item. This formalisation yields a compositional semantics. Finally, the visual query language “visXcerpt” is introduced, showing the advantages of a pattern based approach for the development of a visual query language.

2 Xcerpt Basics

The following principles have prevailed to the definition of the query language: *Pattern-based or positional instead of navigational.* A query should correspond to a form, an answer to a filling yielding a database item. The relative positions of variables in a query should be easily recognisable.

Referential transparency. The meaning of an expression, especially of a variable, should be the same wherever it appears. Therefore, destructive assignments are prohibited and variables must be functional or logic programming variables.

Compositional semantics. A (structurally) recursive definition of the semantics of a query in terms of the semantics of its parts, i.e. a Tarski-style model theory, is sought for.

Multiple variable bindings. Like with SQL and other query languages, queries might have several answers, each answer binding the query variables differently.

Symmetry. Queries should allow similar forms of incomplete specifications in breadth, i.e. concerning siblings, and in depth, i.e. concerning children.

Note that the requirements of [3] are fulfilled by or compatible with the basic query language defined below.

Below, the following pairwise disjoint sets of symbols are referred to: A set \mathcal{I} of identifiers, a set \mathcal{L} of labels (or tags or strings), a set \mathcal{V}_l of label variables, a set \mathcal{V}_t of term (or data item) variables. Identifiers are denoted by id , labels (variables, resp.) by lower (upper, resp.) case letters with or without indices. The following meta-variables (with or without indices and/or superscripts) are used: id denotes an identifier, l denotes a label, L a label variable, X a term variable, t a term (as defined below), v a label or a term, and V a label or term variable.

A database is a set (or multiset) of database terms. The children of a document node may be either ordered (as in standard XML), or unordered. In the following, a term whose root is labelled l and has *ordered* (*unordered*, resp.) children t_1, \dots, t_n is denoted $l[t_1, \dots, t_n]$ ($l\{t_1, \dots, t_n\}$, resp.).

Definition 1 (Database Terms). Database terms are expressions inductively defined as follows and satisfying Conditions 1 and 2 given below:

1. If l is a label, then l is a (atomic) database term.

2. If id is an identifier and t is a database term neither of the form $id_0: t_0$ nor of the form $\uparrow id_0$, then $id: t$ is a database term.
3. If id is an identifier, then $\uparrow id$ is a database term.
4. If l is a label and t_1, \dots, t_n are $n \geq 1$ database terms, then $l[t_1, \dots, t_n]$ and $l\{t_1, \dots, t_n\}$ are database terms.

Condition 1: For a given identifier id an identifier definition $id: t_0$ occurs at most once in a term.

Condition 2: For every identifier reference $\uparrow id$ occurring in a term t an identifier definition $id: t_0$ occurs in t .

Example 1. The following Xcerpt database term describes the book offers of an online book store (This example is inspired from the W3C XQuery Use-Cases [7]).

```

bib {
  a1: author { last{ "Stevens" }, first { "W." } },
  a2: author { last{ "Abiteboul" }, first { "Serge" } },
  a3: author { last{ "Buneman" }, first { "Peter" } },
  a4: author { last{ "Suciu" }, first { "Dan" } },

  book {
    title { "TCP/IP Illustrated" },
    authors [  $\uparrow$  a1 ],
    publisher { "Addison-Wesley" },
    price { "65.95" }
  },
  book {
    title { "Advanced Programming in the Unix environment" },
    authors [  $\uparrow$  a1 ],
    publisher { "Addison-Wesley" },
    price { "65.95" }
  },
  book {
    title { "Data on the Web" },
    authors [  $\uparrow$  a2,  $\uparrow$  a3,  $\uparrow$  a4 ],
    publisher { "Morgan Kaufmann Publishers" },
    price { "39.95" }
  }
}

```

Note that the element order is in general of no importance in a bibliographic database. This is expressed in the Xcerpt syntax using the single curly brackets $\{ \}$. However, in the author list of the first example, order might be of relevance and is thus expressed using the square brackets $[]$. Using the $\uparrow id$ and $id: t$ constructs in the first example make it possible to avoid redundant specifications of the authors. \square

A query term is a pattern that specifies a selection of database terms very much like logical atoms and SQL selections. The evaluation of query terms (cf. Definition 9 on page 8) differs from the evaluation of logical atoms and SQL selections as follows: 1. Answers might have additional subterms to those mentioned in the query term. 2. Answers might have another subterm ordering than the query. 3. A query term might specify subterms at an unspecified depth.

In query terms, the double square and curly brackets, $[[\]]$ and $\{\{ \}\}$, denote exact subterm patterns, i.e. double (square or curly) brackets are used in a query term to be answered by database terms with no more subterms than those given in the query term. $[[\]]$ is used if the subterm order in the answers is to be that of the query term, $\{\{ \}\}$ is used otherwise. Thus, possible answers to the query term $t_1 = a[[b, c\{d, e\}, f]]$ are the database terms $a[b, c\{d, e, g\}, f]$ and $a[b, c\{d, e, g\}, f\{g, h\}]$ and $a[b, c\{d, e\{g, h\}, g\}, f\{g, h\}]$ and $a[b, c\{d, e\}, f]$. In contrast, $a\{b, c\{d, e\}, f, g\}$ and $a[b, c\{d, e\}, f, g]$ and $a\{b, c\{d, e\}, f\}$ are no answers to t_1 .

In a query term, a term variable X can be constrained to some query terms using the construct \rightsquigarrow , read “as”. Thus, the query term $t_2 = a[X_1 \rightsquigarrow b[c, d], X_2, e]$ constrains the term variable X_1 to such database terms that are possible answers to the query term $b[c, d]$. Note that the term variable X_2 is unconstrained in t_2 . Possible answers to t_2 are $a[b[c, d], f, e]$ which binds X_1 to $b[c, d]$ and X_2 to f , $a[b[c, d], f\{g, h\}, e]$ which binds X_1 to $b[c, d]$ and X_2 to $f\{g, h\}$, $a[b[c, d, e], f, e]$ which binds X_1 to $b[c, d, e]$ and X_2 to f , and $a[b[c, e, d], f, e]$ which binds X_1 to $b[c, e, d]$ and X_2 to f . In query terms, the construct *desc*, read “descendant”, specifies a subterm at an unspecified depth. Thus, possible answers to the query term $t_3 = a[X \rightsquigarrow \text{desc } f[c, d], b]$ are $a[f[c, d], b]$ and $a[g[f[c, d]], b]$ and $a[g[f[c, d], h], b]$ and $a[g[g[f[c, d]]], b]$ and $a[g[g[f[c, d], h], i], b]$.

Definition 2 (Query Terms). Query terms are expressions inductively defined as follows and satisfying Conditions 1 and 2 of Definition 1:

1. If l is a label and L is a label variable, then l , L , $l\{\{\}\}$, and $L\{\{\}\}$ are (atomic) query terms.
2. A term variable is a query term.
3. If id is an identifier and t is a query term neither of the form $id_0 : t_0$ nor of the form $\uparrow id_0$, then $id : t$ is a query term.
4. If id is an identifier, then $\uparrow id$ is a query term.
5. If X is a variable and t a query term, then $X \rightsquigarrow t$ is a query term.
6. If X is a variable and t is a query term, then $X \rightsquigarrow \text{desc } t$ is a query term.
7. If l is a label, L a label variable and t_1, \dots, t_n are $n \geq 1$ query terms, then $l[t_1, \dots, t_n]$, $L[t_1, \dots, t_n]$, $l\{t_1, \dots, t_n\}$, $L\{t_1, \dots, t_n\}$, $l[[t_1, \dots, t_n]]$, $L[[t_1, \dots, t_n]]$, $l\{\{t_1, \dots, t_n\}\}$, and $L\{\{t_1, \dots, t_n\}\}$ are query terms.

Query terms in which no variables occur are ground. Query terms that are not of the form $\uparrow id$, are strict. The leftmost label of strict and ground query terms of the form l , $l\{\{\}\}$, $l\{t_1, \dots, t_n\}$, and $l[t_1, \dots, t_n]$ is l ; the leftmost label of a strict and ground query term of the form $id : t$ is the leftmost label of t .

Note that *desc* never occurs in a ground query term, for it is by Definition 2 always coupled with a variable. Database terms are (simple kinds of) query terms. However, the set of answers to a database term t (considered as a query term) in a database D in general contains not only t (cf. Definition 9 on page 8). E.g., f and $f\{a\}$ and $f\{b\}$ are possible answers to f . However, f is the only possible answer to $f\{\}$. In a query term, multiple occurrences of a same term variable are not precluded. E.g. a possible answer to $a\{\{X \rightsquigarrow b\{c\}\}, X \rightsquigarrow$

$b\{\{d\}\}$ is $a\{b\{c, d\}\}$. However, $a[[X \rightsquigarrow b\{\{c\}\}, X \rightsquigarrow f\{\{d\}\}]]$ has no answers, for labels b and f are distinct. *Child subterms* and *subterms* of query terms are defined such that if $t = f[a, g\{Y \rightsquigarrow desc\ b\{X\}, h\{a, X \rightsquigarrow k\{c\}\}\}$, then a and $g\{Y \rightsquigarrow desc\ b\{X\}, h\{a, X \rightsquigarrow k\{c\}\}\}$ are the only child subterms of t and e.g. a and X and $Y \rightsquigarrow desc\ b\{X\}$ and $h\{a, X \rightsquigarrow k\{c\}\}$ and $X \rightsquigarrow k\{c\}$ and t itself are subterms of t . Note that f is not a subterm of t .

Definition 3 (Variable Well-Formed Query Terms). *A term variable X depends on a term variable Y in a query term t if $X \rightsquigarrow t_1$ is a subterm of t and Y is a subterm of t_1 . A query term t is variable well-formed if t contains no term variables X_0, \dots, X_n ($n \geq 1$) such that 1. $X_0 = X_n$ and 2. for all $i = 1, \dots, n$, X_i depends on X_{i-1} in t .*

E.g. $f\{X \rightsquigarrow g\{X\}\}$ and $f\{X \rightsquigarrow g\{Y\}, Y \rightsquigarrow h\{X\}\}$ are not variable well-formed. Variable well-formedness precludes queries specifying infinite answers. In the following, query terms are assumed to be variable well-formed.

Example 2. The following Xcerpt query term is a pattern for the bibliographic database introduced in Example 1 on page 3. An evaluation of this query term against the aforementioned database would yield bindings of TITLE and AUTHOR to all sensible combinations of title/author pairs.

```

bib {{
  book {{
    TITLE ~ title,
    authors { AUTHOR }
  }}
}}
```

□

3 Query Semantics

3.1 Simulation

The semantics is based on graph simulation. Informally, a simulation of a graph G_1 in a graph G_2 is a mapping of the nodes of G_1 in the nodes of G_2 preserving the edges. The graphs considered are directed, ordered and rooted and their nodes are labelled.

Definition 4 (Simulation). *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. Let \sim be an equivalence relation on $V_1 \cup V_2$. A relation $\mathcal{S} \subseteq V_1 \times V_2$ is a simulation with respect to \sim of G_1 in G_2 if:*

1. *If $(v_1, v_2) \in \mathcal{S}$, then $v_1 \sim v_2$.*
2. *If $(v_1, v_2) \in \mathcal{S}$ and $(v_1, v'_1) \in E_1$, then there exists $v'_2 \in V_2$ such that $(v'_1, v'_2) \in \mathcal{S}$ and $(v_2, v'_2) \in E_2$.*

Let \mathcal{S} be simulation \mathcal{S} of $G_1 = (V_1, E_1)$ in $G_2 = (V_2, E_2)$. \mathcal{S} is total, if for each $v_1 \in V_1$ there exists at least one $v_2 \in V_2$ such that $(v_1, v_2) \in \mathcal{S}$. If G_1 has a root r_1 , G_2 has a root r_2 and $(r_1, r_2) \in \mathcal{S}$, then \mathcal{S} is a rooted simulation. \mathcal{S} is minimal, if there are no simulations $\mathcal{S}' \subseteq \mathcal{S}$ of G_1 in G_2 such that $\mathcal{S}' \neq \mathcal{S}$.

Note that every rooted simulation is total.

Definition 5 (Graphs Induced by Strict and Ground Query Terms). Let t be a strict and ground query term. The graph $G_t = (N_t, V_t)$ induced by t is defined by:

1. N_t is the set of strict subterms (cf. Definition 2) of t and each $t' \in N_t$ is labelled with the leftmost label (cf. Definition 2) of t' .
2. V_t is the set of pairs (t_1, t_2) such that either t_2 is a child subterm of t_1 , or $\uparrow id$ is a child subterm of t_1 and the identifier definition $id: t_2$ occurs in t .
3. The children of a node are ordered in G_t like in t .

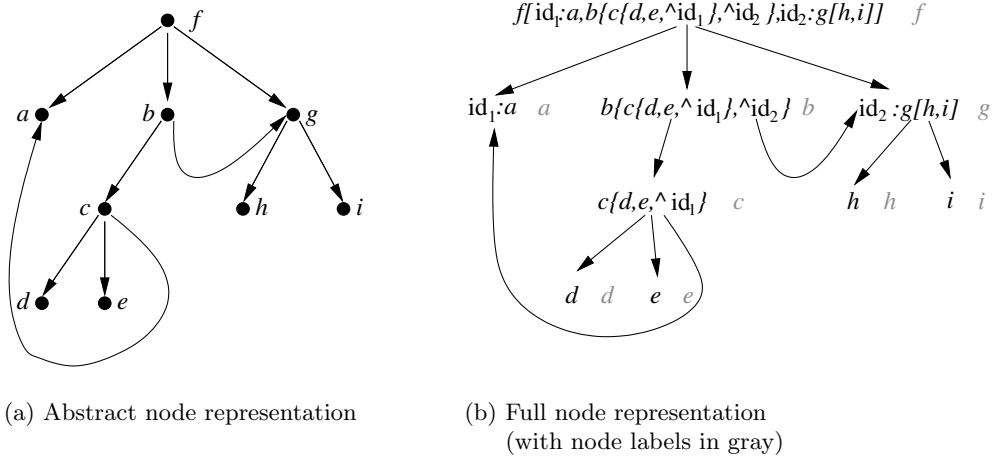


Fig. 1. Graph induced by $t = f[id_1 : a, b\{c\{d, e, \uparrow id_1\}, \uparrow id_2\}, id_2 : g[h, i]]$.

Note that t is the root of G_t . Figure 1 illustrates Definition 5. Obviously, the graph induced by a ground query term does not fully convey the term structure: Missing are graphical representations of the various nestings $[]$, $\{ \}$, $[[]]$ and $\{\{ \}\}$.

Below, a database term is often identified with the graph it induces.

3.2 Simulation Preorder

Definition 6 (Strict and Ground Query Term Simulation). \preceq is the relation on strict and ground query terms defined by $t^1 \preceq t^2$ if there exists a (minimal) rooted simulation with respect to label identity \mathcal{S} of t^1 in t^2 such that:

1. if $v_1 = l\{\}$ occurs in t^1 and $(v_1, v_2) \in \mathcal{S}$, then v_2 has no children in t^2 .
2. if $v_1 = l[[t_1^1, \dots, t_n^1]]$ ($n \geq 1$) occurs in t^1 , $(v_1, v_2) \in \mathcal{S}$ and $(t_i^1, t_j^2) \in \mathcal{S}$ ($1 \leq j \leq m \leq n$), then t_1^2, \dots, t_m^2 occur in this indexing order as children of v_2 in the graph induced by t^2 .

3. if $v_1 = l[t_1^1, \dots, t_n^1]$ ($n \geq 1$) occurs in t^1 , $(v_1, v_2) \in \mathcal{S}$ and if $(t_i^1, t_j^2) \in \mathcal{S}$ ($1 \leq j \leq m \leq n$), then t_1^2, \dots, t_m^2 are pairwise distinct (i.e. $m = n$), they occur in this indexing order as children of v_2 in the graph induced by t^2 and v_2 has no other children than the t_j^2 in t^2 .
4. if $v_1 = l\{t_1^1, \dots, t_n^1\}$ occurs in t^1 , $(v_1, v_2) \in \mathcal{S}$ and $(t_i^1, t_j^2) \in \mathcal{S}$ ($1 \leq j \leq m \leq n$), then v_2 has no other children than the t_j^2 in t^2 .

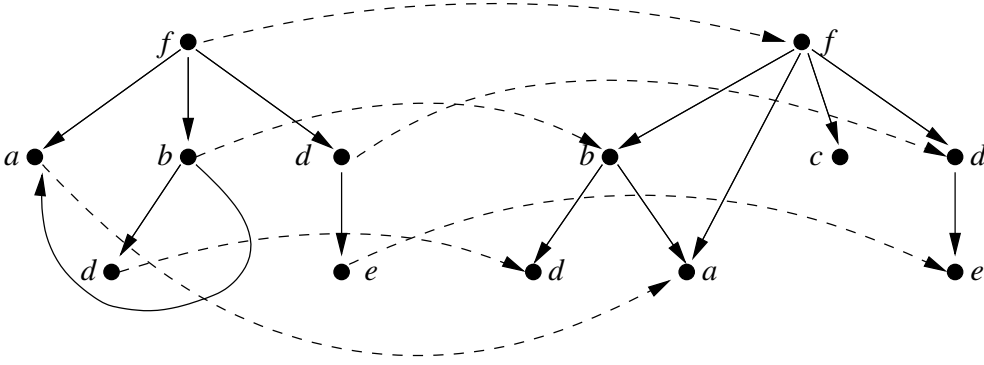


Fig. 2. A minimal simulation of the (graph induced by the) ground query term $t^a = f\{\text{id}_1 : a, b[d\{\{\}\}], \uparrow \text{id}_1\}$, *desce* in the (graph induced by the) database term $t^{db} = f[b[d, \text{id}_2 : a], \uparrow \text{id}_2, c, d\{e\}]$.

Figure 2 illustrates Definition 6.

By Definition 4, \preceq is reflexive and transitive, i.e. it is a preorder on the set of database terms.

\preceq is not a partial order, for although $t_1 = f\{\{a\}\} \preceq t_2 = f\{\{a, a\}\}$ and $t_2 = f\{\{a, a\}\} \preceq t_1 = f\{\{a\}\}$ (both a of t_2 can be simulated by the same a of t_1), $t_1 = f\{\{a\}\} \neq t_2 = f\{\{a, a\}\}$.

3.3 Grounding Substitutions and Ground Instances

Rooted simulation with respect to label equality is a first notion towards a formalisation of answers to query terms: If there exists a rooted simulation of (the graph induced by) a database term t_1 , considered as a query term, in (the graph induced by) a database term t_2 , then t_2 is an answer to t_1 . Ground instances of a query term (cf. Definition 7 on the next page) gives rise to extend this notion of answers to query terms that are not database terms. An answer in a database D to a query term t^a is characterised by bindings for the variables in t^a such that the database term t resulting from applying these bindings to t^a is an element of D . Consider e.g. the query $t^a = f\{\{X \rightsquigarrow g\{b\}\}, X \rightsquigarrow g\{c\}\}$ against the database $D = \{f\{g\{a, b, c\}, g\{a, b, c\}, h\}, f\{g\{b\}, g\{c\}\}\}$. The \rightsquigarrow constructs in t^a yield the constraint $g\{b\} \preceq X \wedge g\{c\} \preceq X$. Matching t^a with the first database term in D yields the constraint $X \preceq g\{a, b, c\}$. Matching t^a with the second database term in D yields the constraint $X \preceq g\{b\} \wedge X \preceq g\{c\}$. $g\{b\} \preceq X \wedge g\{c\} \preceq X$ is not compatible with $X \preceq g\{b\} \wedge X \preceq g\{c\}$. Thus, the

only possible value for X is $g\{a, b, c\}$, i.e. the only possible answer to t^q in D is $f\{g\{a, b, c\}, g\{a, b, c\}, h\}$.

Definition 7 (Ground Instances of Query Terms). A grounding substitution is a function which assigns a label to each label variable and a database term to each term variable of a finite set of (label or term) variables. Let t^q be a query term, V_1, \dots, V_n be the (label or term) variables occurring in t^q and σ be a grounding substitution assigning v_i to V_i . The ground instance $t^q\sigma$ of t^q with respect to σ is the ground query term that can be constructed from t^q as follows:

1. Replace each subterm $X \rightsquigarrow t$ by X .
2. Replace each occurrence of V_i by v_i ($1 \leq i \leq n$).

Requiring in Definition 2 *desc* to occur to the right of \rightsquigarrow makes it possible to characterise a ground instance of a query term by a grounding substitution. This is helpful for formalising answers but not necessary for language implementations.

3.4 Answers

Not all ground instances of a query term are acceptable answers, for some instances might violate the conditions expressed by the \rightsquigarrow and *desc* constructs.

Definition 8 (Allowed Instances). The constraint induced by a query term t^q and a substitution σ is the conjunction of all inequalities $t\sigma \preceq X\sigma$ such that $X \rightsquigarrow t$ is a subterm of t^q not of the form *desc* t_0 , and of all expressions $X\sigma \triangleleft t\sigma$ (read “ $t\sigma$ subterm of $X\sigma$ ”) such that $X \rightsquigarrow \text{desc } t$ is a subterm of t^q , if t^q has such subterms. If t^q has no such subterms, the constraint induced t^q and σ is the formula true. Let σ be a grounding substitution and $t^q\sigma$ a ground instance of t^q . $t^q\sigma$ is allowed if:

1. Each inequality $t_1 \preceq t_2$ in the constraint induced by t^q and σ is satisfied.
2. For each $t_1 \triangleleft t_2$ in the constraint induced by t^q and σ , t_2 is a subterm of t_1 .

Definition 9 (Answers). Let t^q be a query term and D a database. An answer to t^q in D is a database term $t^{db} \in D$ such that there exists an allowed ground instance t of t^q satisfying $t \preceq t^{db}$.

4 Basics of the visual query and transformation language visXcerpt

visXcerpt is a visual language fully based on Xcerpt. It covers all aspects of Xcerpt and does not introduce elements which can not be mapped directly to Xcerpt elements. Therefore, Xcerpt programs can be translated into visXcerpt programs and vice versa without information loss.

Database terms are represented in visXcerpt through nested rectangular boxes with attached “tabs”. This reminds somehow of the tabs used in graphical windowing toolkits, which supports the awareness of an underlying hierarchical structure. The child-elements are nested inside the box. Because visXcerpt is primarily intended to be used with XML, there is special handling of textual content (so called CDATA), attributes and namespaces¹ (all of which are not part of the data model introduced above): Attributes are placed in a two-column table with the name in the left column and the value in the right one. The attribute table is the first content of the box. If there are no attributes, the table is omitted. Sub-terms (i.e. child nodes), are recursively visualised the same way and arranged vertically inside of the parent element’s box. Textual content is simply written as plain text. In visXcerpt the namespace of an element is written on the top right corner in a smaller font and attribute namespaces are prefixed (also in smaller font) before the attribute name inside it’s table cell. For better distinction of elements at different levels, every level is coloured differently.

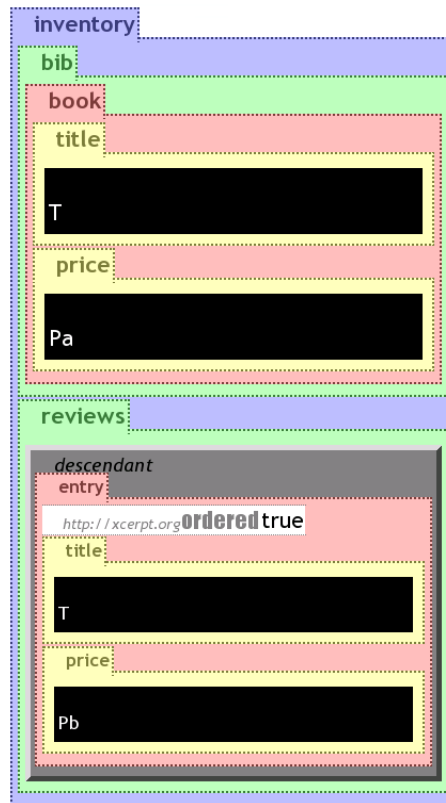


Fig. 3. Query for the prices of the same books in two different data stores represented by the elements `bib` and `reviews`.

XML allows the construction of graphs using reference mechanisms. Graph representation is not fully implemented in visXcerpt and a matter of ongoing work. The solution retained for visXcerpt is the use of hyperlinks to cope with graph structure – the graph is then represented by a spanning tree and references used to complete the graph are spanned in the “hyperspace”. This departs from the approaches taken in most other visual languages, which are using two dimensional representations of nodes and edges for graph visualisation. While a two dimensional visualising non-tree graphs is suitable for small graphs, a hyperlink representation scales very well to graphs of arbitrary sizes. A reference is visualised as a link (see 1b in Figure 4 on the following page) to the occurrence of the referenced element (1a in Figure 4). Clicking onto such a link (see 3a) scrolls the user’s visual context to the referenced element (3b). Furthermore, when hovering with the mouse pointer over a referenced element (see 2a), all referring elements are highlighted (2b). While browsing or editing documents using visXcerpt, it is also possible to navigate back to the referencing element via a back referring link (not shown).

¹ Namespaces are defined as W3C recommendation at <http://www.w3.org/TR/REC-xml-names/>

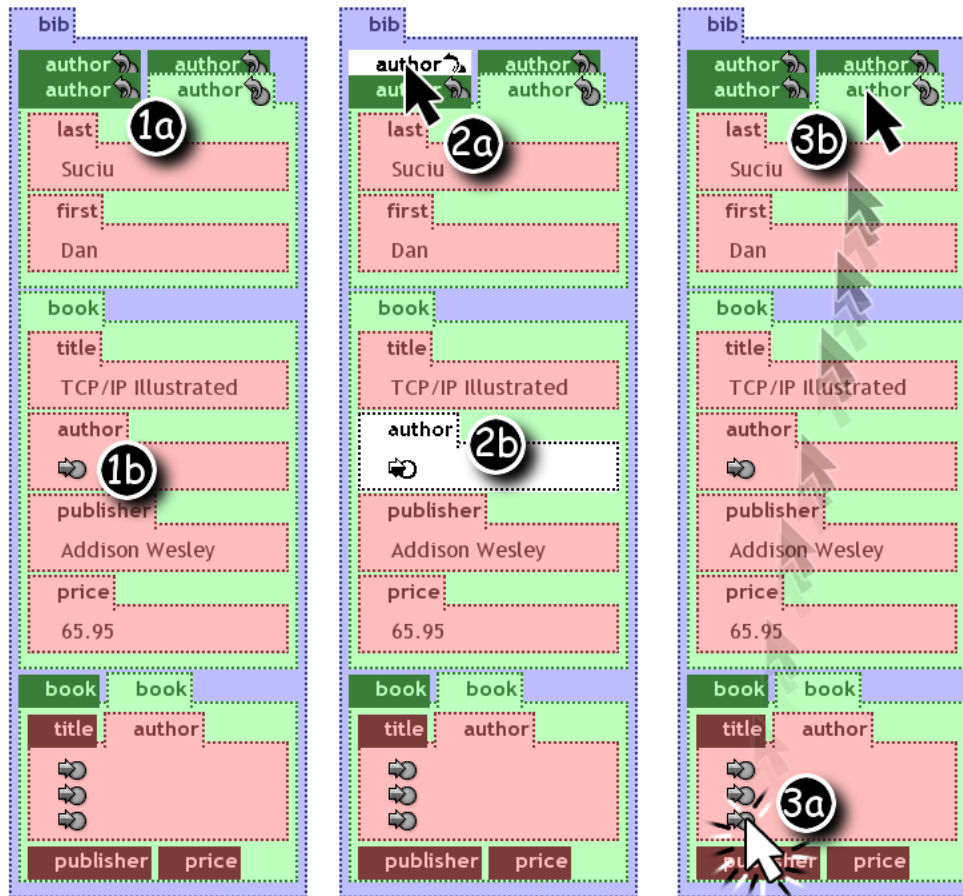


Fig. 4. Proposed handling of graph structures. Clicking on a reference (3a) scrolls the users visual context to the referenced element (3b).

To cope with Xcerpt query terms, the following mapping of query constructs is proposed:

- **term variables** : Term variables are represented as black boxes with their name in the left top corner (see the box called VAR1 in Figure 5 on the next page). If apart of the variable name the content of that box is empty, the variable is considered to be unconstrained. If the variable is constrained by an “as”-construct, the constraining pattern is the content of the black variable box. This emphasises that all objects “collected” by the variable (and therefore inside of the box) resemble to the pattern.
- **the descendant construct** : Descendants are visualised as grey boxes with the word “descendant” written in the left top corner in italic. Italic font style is reserved to textual Xcerpt key words that are used to support better association with the textual representation. Xcerpt keywords are intended to be used when no other text like a variable or element name is involved, and when the construct is used infrequently. The metaphor of three dimensional *depth* is used through a bevelled border to indicate matching of the contained pattern at arbitrary *depth*.

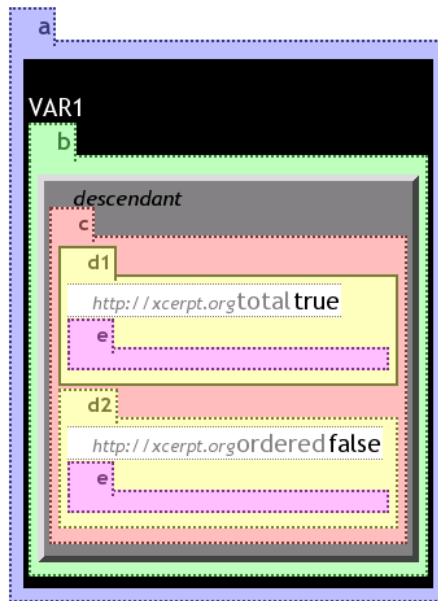


Fig. 5. Example pattern with variable, descendant construct, ordered/unordered and total/partial matched elements.

- **exact and partial matching :** The border of visXcerpt elements are used to indicate this feature (see the box called `d1` in Figure 5). If the border is painted as a solid line, the element’s content is intended to be matched exactly, if the line is dotted or dashed partial matching of the child elements is indicated.
- **ordered and unordered matching :** This feature is currently not visually supported, but an XML attribute named `ordered` in the namespace `http://xcerpt.org/` can be set to `true` or `false` leading to the desired effect. Hence, the generic attribute visualisation is used. In future versions of visXcerpt, different alignment may be used to represent unordered matching: while ordered content is represented using left text alignment, unordered content may be aligned centred. This affects the tabs and the textual content.

4.1 Interactivity

An important property of a visual programming language is an editor which is capable of supporting the user with both browsing and intelligent editing facilities, especially when editing complex programs or documents.

Browsing Nodes can be folded by clicking on their tab. The content is hidden while light and dark shading of colours used in the tab are reversed. The labels are then displayed like tabs on registers along with other folded sibling elements, an unfolded element label is the last one in the row of labels (see Figure 6 on the next page). If the amount of folded labels exceeds the width of the visXcerpt

display area, a line-break occurs and a new line begins. All nodes can be folded in the initial state, which is important for large databases. The user can then dig into relevant areas by iterative unfolding of relevant nodes.

Editing The approach described so far only covers handling of static data. To be able to create or edit visXcerpt documents, it is important to have an editor model that describes state transitions the user can perform in an interactive environment for visXcerpt. While plain text as attribute values and text nodes can be edited the usual way, editing models for tree or graph structures are not yet established.

The editing paradigm used in visXcerpt is the copy-and-paste paradigm. Each element has a context menu that contains appropriate operations. Copying an element results in copying an element with all its child nodes into a buffer. Copying an element is unambiguous, but the term 'paste' does not state clearly where to paste something. Therefore paste is refined in the four operations

- **Paste before:** The buffer's content is pasted directly before the content element.
- **Paste after:** The buffer's content is pasted directly after the content element.
- **Paste into at beginning:** The buffer's content is pasted into the content element as first child.
- **Paste into at end:** The buffer's content is pasted into the content element as last child.

Further editing operations acting on a context element are **cut** and **delete**, where **cut** behaves like first copying and then deleting the content element. The **delete**-operation removes a node with all its child nodes.

The former editing operations are explained with tree structures in mind which is apparent, when "child elements" are mentioned. Because graph structures in visXcerpt are represented by hyperlinks on spanning trees, the handling of graph structures is similar to the handling of trees. Nevertheless, graph specific extensions need to be investigated. A graph specific extension could be to check referential consistency, maybe by removing all relevant references when an element is deleted.

5 Concluding Remarks

The articles [8,9,10] have already pointed out the drawbacks of relying on a navigational node selection à la XPath [11] and XQuery [12] for query and transformation languages for XML and semistructured data.



Fig. 6. Example with folded content (the tabs labelled "reviews", "price", "authors" and "ISBN") and context menu button (at the tab labelled "title")

[9] describes a language called *fxt* that has variables for terms, corresponding to trees, and forests. In *fxt*, node selection is done with regular expressions. In contrast to *Xcerpt* and the basic language described above uses term variables for this purpose. In [8], a query and transformation language is described that is related to logic and Prolog. This language has (in the terminology used above) only label variables. In contrast, the basic query language introduced above also has term variables.

The article [13] describes a non-standard unification algorithm subjacent to Definition 9. [14,15] introduce in more detail the query language *Xcerpt* which builds upon the basic language constructs introduced above. *Xcerpt* has construct terms in which variables but no *desc* and \rightsquigarrow may occur. A construct term with variables V_1, \dots, V_n is associated with a conjunction or disjunction of (possibly negated) query terms in which all of V_1, \dots, V_n occur. *Xcerpt* has additional features that for space reasons cannot be mentioned here. A prototype has been realized that implements a set-oriented backward reasoning operational semantics. First experiments suggest that the *Xcerpt* approach to querying is convenient in practice.

The language UnQL [5] has introduced simulation as a means for query answering. UnQL, like *Xcerpt*, uses the notions of patterns and templates. UnQL and *Xcerpt* differ from each other as follows. First, a query in UnQL consists of a single “select-where” expression which can be processed with pattern matching. In contrast, a query in *Xcerpt* might “chain” several “construct-query rules” requiring a “unification” which is capable of binding variables from both of the terms to be “unified”. Second, variables in UnQL can only occur as leaves of query patterns. Complex queries might require the use of several patterns in UnQL, where a single pattern suffices in *Xcerpt*.

References

1. Henzinger, M.R., Henzinger, T.A., Kopke, P.W.: Computing Simulations on Finite and Infinite Graphs. Technical report, Computer Science Department, Cornell University (1996)
2. World Wide Web Consortium (W3C) <http://www.w3.org/>. (2002)
3. Maier, D.: Database Desiderata for an XML Query Language. In: Proceedings of QL'98 - The Query Languages Workshop. (1998) <http://www.w3.org/TandS/QL/QL98/>.
4. Olteanu, D., Meuss, H., Furche, T., Bry, F.: XPath: Looking Forward. In: Proceedings of Workshop on XML Data Management (XMLDM), <http://www.pms.informatik.uni-muenchen.de/publikationen/#PMS-FB-2002-4>, Springer-Verlag LNCS (2002)
5. Buneman, P., Fernandez, M., Suciu, D.: UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion. *VLDB Journal* **9** (2000) 76–110
6. Baru, C., Ludöscher, B., Papakonstantinou, Y., Velikhov, P., Vianu, V.: Features and Requirements for an XML View Definition Language: Lessons from XML Information Mediation. In: QL98/W3C. (1998)
7. Chamberlin, D., Fankhauser, P., Marchiori, M., Robie, J.: XML query use cases. W3C Working Draft 20 (2001)
8. Grahne, G., Lakshmanan, L.V.S.: On the Difference between Navigating Semi-structured Data and Querying It. In: Proceedings of Workshop on Database Programming Languages. (1999) 271–296
9. Berlea, A., Seidl, H.: *fxt* – A Transformation Language for XML Documents. *J. of Computing and Information Technology*, Special Issue on Domain-Specific Languages (2001)

10. Boley, H.: Relationships between logic programming and XML. In: Proceedings of 14th Workshop Logische Programmierung, Würzburg (2000)
11. W3 Consortium <http://www.w3.org/TR/xpath>: XML Path Language (XPath). (1999)
12. W3C <http://www.w3.org/TR/xquery/>: XQuery: A Query Language for XML. (2001)
13. Bry, F., Schaffert, S.: Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. In: Proceedings of the Int. Conf. on Logic Programming (ICLP), Copenhagen, Springer-Verlag LNCS (2002)
14. Bry, F., Schaffert, S.: A Gentle Introduction into Xcerpt, a Rule-based Query and Transformation Language for XML. In: Proceedings of the International Workshop on Rule Markup Languages for Business Rules on the Semantic Web, Sardinia, Italy (2002) (invited article).
15. Bry, F., Schaffert, S.: The XML Query Language Xcerpt: Design Principles, Examples, and Semantics. In: Proceedings of the 2nd Annual International Workshop "Web and Databases", Erfurt, Germany (2002)

